INFO-F-101 Programmation

Projet n° 6

Année académique 2004–2005

Dans le cadre de ce projet, nous vous demandons d'écrire des fonctions permettant de gérer un carnet d'adresses, tel qu'on en trouve aujourd'hui sur tous les téléphones mobiles. Le carnet d'adresses sera stocké sous forme d'une liste *triée*, *doublement liée*, *circulaire* et avec *élément pré-tête*. Chaque élément de la liste représentera un contact, pour lequel diverses informations seront stockées : le nom, le prénom, différents numéros de téléphone, et l'adresse.

1 Déclarations

Dans le cadre de ce projet, nous considérerons les déclarations suivantes :

```
const int LgChaine = 200 ;
const int LgNum = 15 ;

typedef char chaine[LgChaine] ;
typedef char numero[LgNum] ;

class elem ;

typedef elem * carnet ;

class elem {       // Les éléments de la liste
 public:
  chaine nom ;     // Nom de famille
  chaine prenom ;  // Prénom
  numero maison ;  // Numéro de téléphone domicile
  numero mobile ;  // Numéro de téléphone mobile
  chaine adresse ; // Adresse

  carnet next ;    // Elément suivant dans la liste
  carnet prev ;    // Elément précédent
  elem() ;
  elem(char _nom[], char _prenom[],
       char _maison[], char _mobile[],
       char _adresse[], carnet _next, carnet _prev) ;

} ;
```

Ces déclarations sont disponibles sous forme d'un fichier `carnet.h`, sur la page des TPs. Vous *devez impérativement* vous servir de ce fichier dans votre projet (*via* un `#include`). Le code des constructeurs vous est également fourni dans un fichier `carnet.cpp`.

Nous manipulerons toujours des listes doublement liées et circulaires d'éléments de type `elem`, avec élément pré-tête. De plus, ces listes devront être triées sur base du champ `nom` des éléments.

On accédera toujours à ces listes *via* un pointeur vers l'élément pré-tête. Ce pointeur sera donc de type `carnet`.

2 Fonctions demandées

On vous demande de réaliser les fonction suivantes :

1. `void init(carnet &c)` qui initialise la liste pointée par `c` en créant une liste vide.
2. `bool PPQ(chaine s1, chaine s2)` cette fonction retourne *vrai* si la chaîne `s1` apparaît *strictement avant* la chaîne `s2` dans l'ordre lexicographique (il s'agit de l'ordre utilisé dans un dictionnaire : dans le cadre de ce projet, vous pouvez faire l'hypothèse que les noms ne sont composés que de caractères alphabétiques en majuscule). Elle retourne *faux* dans le cas contraire. **Exemples :** `PPQ(''ABC'', ''CDE'')` retourne *vrai*, `PPQ(''ZXY'', ''ABC'')` retourne faux, `PPQ(''ABC'', ''ABC'')` retourne faux et `PPQ`("PAUL", "PAULINE")- retourne *vrai*.
3. `EGAL(chaine s1, chaine s2)` cette fonction retourne vrai les deux chaînes `s1` et `s2` sont identiques. Elle retourne *faux* dans le cas contraire.
4. `void insertion(elem * e, carnet c)` cette fonction doit insérer l'élément pointé par `e` dans la liste pointée par `c`. L'insertion doit se faire de façon triée selon l'ordre lexicographique sur base du champ `nom`. Dans le cas où il existe déjà dans `c` un élément (que nous appelerons `f`) tel que `f.nom` = `e.nom` et `f.prenom` = `e.prenom`, la fonction fusionne les deux éléments. Pour ce faire, elle crée un nouvel élément contenant l'information contenue dans `e` et `f`. En cas de conflit (par exemple si l'adresse est spécifiée dans `e` et `f`), la fonction demande à l'utilisateur de choisir quelle information il y a lieu de conserver. Cette fonction doit tirer parti de `PPQ` et `EGAL`.
5. `void affichage(chaine nom, chaine prenom, carnet c)` cette fonction recherche dans `c` le contact dont les nom et prénom sont spécifiés, et l'affiche à l'écran. À nouveau, vous devez utiliser les fonction `PPQ` et `EGAL`.
6. `void fusion(carnet c1, carnet c2)` cette fonction réalise la fusion de `c2` et `c1` et met le résultat dans `c1` (la liste `c2` reste inchangée). En cas de doublon, on procéde comme dans le cas de l'insertion. Il est conseillé de faire appel à la fonction `insertion` pour plus de modularité. On vous demande d'optimiser cette fonction avec soin. En particulier, essayerz, de ne pas devoir parcourir la liste `c1` à partir du début lors de chaque insertion.

Enfin, on vous demande un `main` démontrant le fonctionnement de vos fonctions. Celui-ci créera, par exemple, deux listes, fera quelques insertions puis une fusion.

3 Exemple de fusion

Pour reprendre notre exemple de téléphone portable, imaginons que Farid possède dans son téléphone la liste de contact `c2`, et qu'il l'envoie par connexion Bluetooth à Thierry, dont le téléphone contient la liste `c1` (voir Fig. 1).

Le téléphone de Thierry va alors effectuer le fusion des deux listes, en appelant `fusion(c1, c2)`. Comme l'entrée *Sherlock Holmes* est présente dans les deux listes, un message de ce type sera affiché :

```
L'entrée SHERLOCK HOLMES existe déjà.
Choissisez l'information à conserver (domicile):
1 - 004478514112
2 - 024715241
```

En supposant que Thierry choisisse l'option 1, sa liste de contacts se présentera comme montré à la Fig. 2.

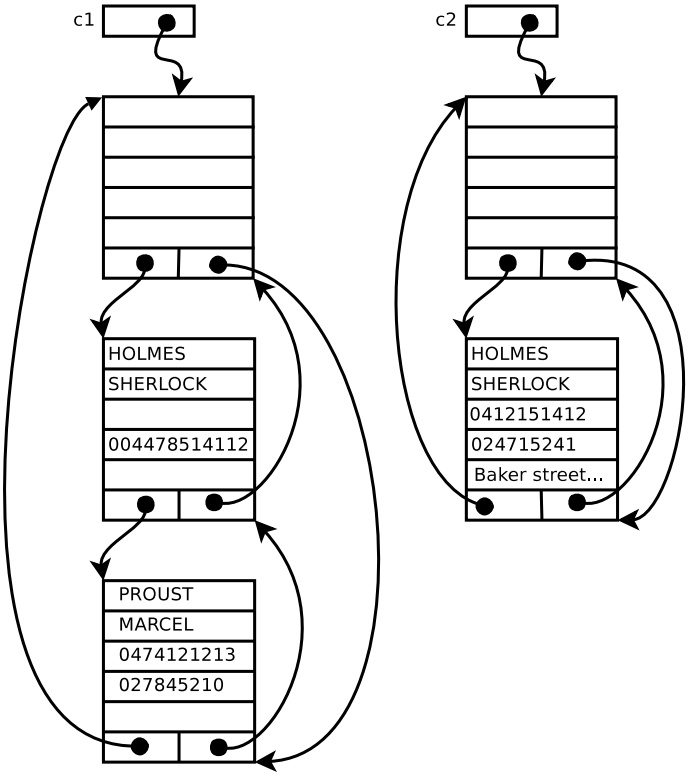

FIG. 1 – Les deux listes avant la fusion.

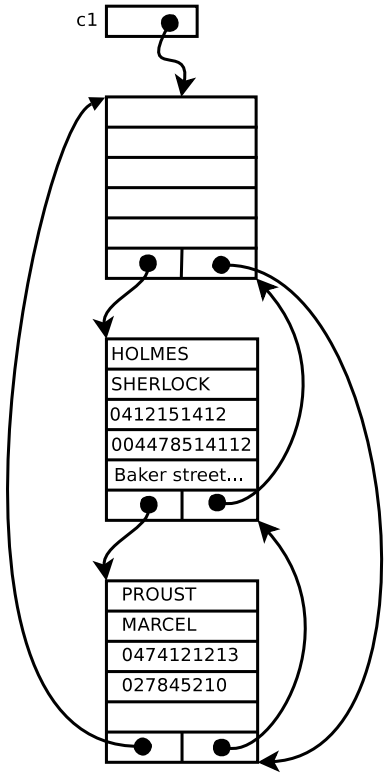

FIG. 2 – La liste `c1` après la fusion.

4 Consignes

On vous demande de remettre le code *C++* du programme (sous format papier), intelligemment commenté, au secrétariat étudiant du département d'informatique (2N8.104), le 17 décembre, au plus tard à 16h (un point en moins par jour de retard). Pensez à préciser le nom de votre assistant sur la première page. On vous demande également un *rapport* décrivant les détails de votre implémentation[1]. Enfin, vous devrez rendre disponible le code source de votre projet (c'est-à-dire tous les fichiers nécessaires à sa compilation) dans un sous-répertoire `projet6` (en *minuscules*) de votre *home* sur `di-net` (salles PC du 4^{e}étage). Réfèrez-vous au mode d'emploi des salles pour savoir comment créer un répertoire et y copier vos fichiers. Le mode d'emploi a été distribué au début de l'année, et est encore disponible à l'adresse :
`http://www.ulb.ac.be/di/ssd/madewulf/cours/INFO-F-101/ModeEmploi.pdf`

Après avoir créé le répertoire et copié les fichiers, tapez les deux commandes suivantes :

```
cd
chmod -R o+r projet6/
```

Elles ont pour effet de permettre à vos assistants de lire vos fichiers, et sont donc indispensables. Par contre, cette modification ne permet pas aux autres étudiants d'accéder à votre projet.

[1] Voir le document `http ://www.ulb.ac.be/di/ssd/tmassart/API/rapport.html`.